COMPARISON OF SUPPORT STRUCTURE GENERATION TECHNIQUES FOR 3D PRINTING

By
Ann Tupek

RECOMMENDED:

Dr. Orion Lawlor, Committee Chair

Dr. Glenn Chappell, Committee Member

Dr. Chris Hartman, Committee Member

Comparison of Support Structure Generation Techniques for 3D Printing

By Ann Tupek

Graduate Advisory Committee:
Chair: Dr. Orion Lawlor
Members: Dr. Glenn Chappell
Dr. Chris Hartman

M.S. Project Report

Presented to the faculty of the University of Alaska Fairbanks in partial fulfillment of the requirements of Master of Science in Computer Science, 2016.

Abstract

As the old patents from the late 1980's and early 1990's expire, additive fabrication and rapid prototyping has seen a boom in recent years. Often called 3D printing, in the additive fabrication process material is deposited on previous layers. The problem of supporting overhangs in 3D printing is not one encountered in the traditional method of subtractive fabrication. This paper examines and compares three methods of generating support structures for these overhangs: a scaffolding structure, simple pillars, and the supports automatically generated by the open source slicing tool Slic3r. Our results show that both the scaffolding structure and simple pillars use less material than Slic3r's supports. Additionally, the scaffolding structure and simple pillars print in a comparable amount of time as Slic3r's supports and all the models have a similar visual print quality. This conservation of material without a reduction in print quality make our method of scaffolding support structures preferable to the supports automatically generated by Slic3r.

Table of Contents

Introduction

There are many different methods of 3D printing, including: stereolithography (SLA), selective laser sintering (SLS), laminated object manufacturing (LOM), and Fused Deposition Modeling (FDM) or Fused Filament Fabrication (FFF). FFF is a 3D printing method in which plastic filament is melted and then extruded through a nozzle and deposited onto the layer below it. Consumer 3D printers are frequently FFF printers, though some SLA models are available.

An advantage of additive fabrication over subtractive fabrication is that less material can be used. Instead of carving away or removing material, which results in waste that can often not be reclaimed, additive fabrication builds up on the previous layer. It also allows for building complex shapes and intricate internal structures that would be impossible to machine from one piece using subtractive fabrication and decreases the amount of operation time spent by humans to build parts [1].

In the method of subtractive fabrication, overhangs, material with nothing beneath it, do not require support, as material is removed from beneath them as the model is created. However, overhangs present a problem for additive fabrication, as there is nothing beneath them for the next layer of material to be deposited or fused to. Therefore, the successful printing of overhangs requires some support structure that can be removed from the model during a post-processing phase.

There are several distinct methods of generating such support material, and each method has its own advantages and disadvantages. The method described in this paper is one that produces a support structure scaffolding of bridges and pillars that support overhangs while minimizing the amount of filament used to create the support structure.

We then compare this scaffolding support structure to a simpler support structure consisting of straight pillars and a support structure generated automatically by the open-source slicing tool Slic3r (version 1.2.9).

Prior Work

Determining Points that Require Support

There are two methods of determining which points in a model require support. In the first one, the downward-facing triangles of the .stl file are considered. The steepness of their normal vectors determines whether or not they need support. The second method computes the Boolean difference between successive layers in a sliced model. The points that are present in the layer above, but absent in the layer below are those that require support. This paper uses the Boolean difference method to determine which points in a model to support.

Support Structures

Dense Pillars

In this method, dense pillars or walls are dropped straight down from the points requiring support. This results in the use of a lot of material that must be removed from the object in post processing, and increases the time spent post processing the model as the interface layer between the model and supports is generally dense and difficult to remove without damaging the item. However, this method, if used with a dissolvable support material such as PVA on a dual-extrusion machine, can result in very clean prints.

Sparse Pillars

Sparse pillars are similar to dense pillars, however the points that need support are down-sampled and pillars are dropped down from the resulting points to support the model's overhangs, resulting in less material used to support the model's overhangs. This is the method we use for our simple support method for comparison to our scaffold support method.

Lattice Structures

Lattice support structures are based on a cell geometry. The individual cells are replicated to fill the space under the model that needs support. Hussein, et al found that cells with small volume fractions, that is the percent of solid volume compared to the open volume in the cell, are capable of supporting overhangs in a model and reducing curl and distortion [3]. Strano, et al also employed cellular structures to support overhangs in models [5]. Zein, et al developed a porous honeycomb-like structure for use in tissue engineering [7]. This lattice structure method also allows for the ease of recovering raw loose powder trapped inside support structures during the build which makes it ideal for SLS methods.

Scaffold & Tree Structures

Scaffold and tree structures attempt to support points by growing a support structure down from the points that need support. These complex structures attempt to minimize the amount of material used by grouping supports into tree-like structures or scaffolding similar to that used in building construction and show

that sparse structures can be used for supports. Vanek, et al. demonstrated that their tree-like structure used less material than dense pillar support structures [6]. The scaffolding structure takes advantage of a 3D printer's ability to print short bridges which are supported at its endpoints, but not from beneath. Dumas et al. describe a method of generating a scaffolding support structure and demonstrate that the bridge structure is more efficient than a tree-like structure and more stable during the printing process [2]. Their method is the one implemented in this paper.

Our Method

The 3D Printing Process

After a model is sourced, whether designed by the user or downloaded from a website such as Thingiverse.com, it must be translated into code that can be read by the 3D printer. These models typically are available in STL format. A STL file is a mesh of connected triangles that represent the different facets of the model's surface, but it is not generally a file that can be read by a 3D printer, and so it must be translated into code that the machine can follow. This translation process is called *slicing* and it creates horizontal cross sections of the model and from those cross sections creates the paths that the print head follows in the form of gcode. The gcode is then loaded into the printer, which then prints the model.

Assumptions

To minimize the overhangs that need support, we assume that the user has oriented the model in an optimum configuration within the slicer. We also assume some standard slicer settings: a 0.2mm layer height, 0.5mm nozzle diameter. The models printed in this experiment were printed on a RepRap Prusa I3 at a speed of 40mm/s.

Our Process

We begin with an STL file of our model. We then slice it into gcode using Slic3r with its support generation turned off. We parse through the gcode to determine the points to support. Those points go into out support structure generation tool which constructs the bridges and pillars and outputs them into an OpenSCAD file. OpenSCAD is an open source software program for creating 3D CAD models. Our model is then unioned with the bridges and pillars that support it. We then render the model in OpenSCAD and export it as an STL file. This STL file is then sliced and the resulting gcode is given to the 3D printer. The model is then printed.

Generating Supports

For our method, we first slice the STL file into gcode using Slic3r without support generation. To get the points that need support, we do a Boolean difference of gcode layers. The gcode lists the endpoints of the movements of the print head as it deposits filament. We only look at the G1 move commands where filament is actually extruded from the nozzle. We put these points into a 2D vector of pixels at each layer. Using a modified Bresenham Algorithm, lines are drawn between subsequent endpoints. This approximates the deposition of filament between the

endpoints. We then compare each layer to the one beneath it. Any point without support will show up in the difference.

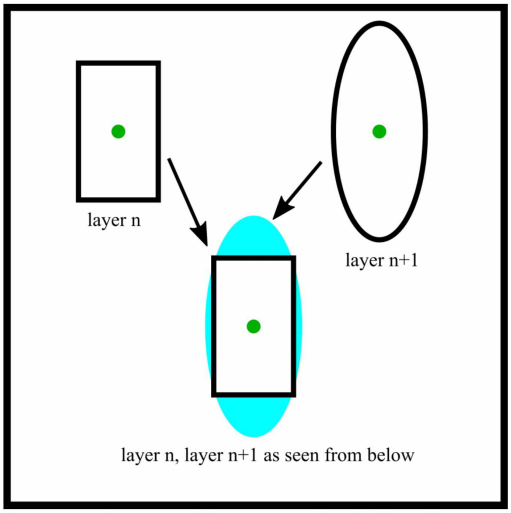

Figure 1 Boolean Difference Between Layers. The green dot is for alignment. The blue area is the section of layer n+1 that is not supported from below.

We then down-sample these points by overlaying a coarser-resolution grid on the points that need support to maintain a minimum distance of 5mm between each pillar. The points derived from the coarser grid are the points that require support and are output to a file. For our simple support method, a straight pillar is dropped down from each of these points to create the model's support structure.

To create the scaffolding, the points that require support are read in from the file. The set of active points is initialized with these points. Following the algorithms described in Dumas, et al, several sweep planes are produced [2]. These are at angles of 0, 22.5, 45, 67.5, 90, 112.5, 135, and 157.5 degrees from the x-axis (multiples of pi/8 radians). Anchoring segments perpendicular to each sweep plane are created at each point that requires support. We then find the places at which the sweep plane intersects the anchoring segments. These intersection points are used to generate temporary bridges.

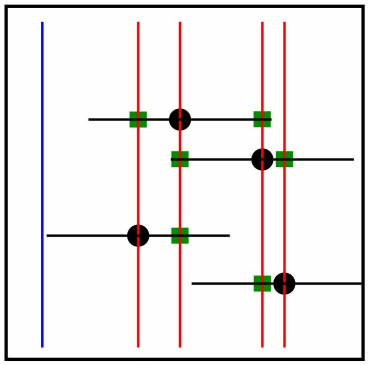

Figure 2 Sweep Plane: The black dots represent the points to be supported, as projected onto the xy-plane. The black lines are the anchoring segments. The blue line is the sweep plane, and the red lines represent the sweep plane as it reaches each point. The green squares are the intersection points. Along with the points that need support, these green squares will be considered as the endpoints for potential bridges.

Each temporary bridge is evaluated to determine its *gain*. Gain is calculated as (k-2)*height-length, where k is the number of points supported by the bridge. A temporary bridge is only further considered if its gain is positive, so we only look at bridges that support at least three points. The l_max of a positive-gain bridge is computed as the maximum horizontal and vertical distance to a point that it supports. The gain and l_max are then used in the computation to determine the temporary bridge's score. The bridge's score is the gain - (k * l_max), and is allowed to be negative.

Points are considered supported by the bridge if they are at least 2mm above the bridge and within 5mm of horizontal distance from the bridge. This allows for situations in which points are offset from the bridge and do not line up directly. These constraints were selected to ensure the slanted pillars between the bridge and the model print correctly. Future work would involve optimizing these constraints and finding the threshold at which they break down.

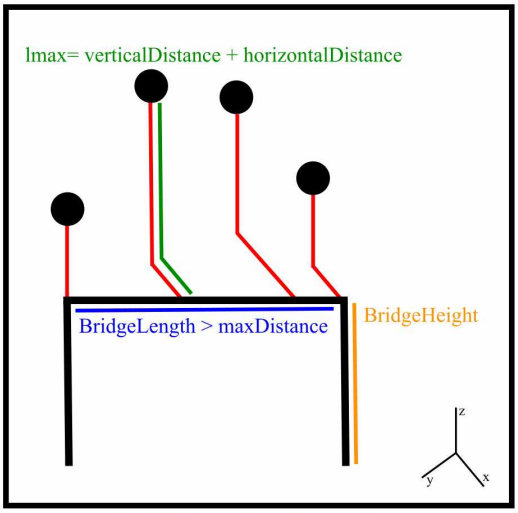

Figure 3: BridgeDistance, BridgeHeight, and lmax. The black dots are the points that require support and the black lines represent the bridge being evaluated.

At each iteration of the algorithm, the temporary bridge with the best score becomes the best bridge. In the snap function, the points supported by the best bridge are removed from the set of active points, and the bridge's endpoints are added to the set of active points. Additionally, the OpenSCAD components of the bridge are created. A slanted pillar is dropped down from each supported point to the bridge and a bridge surface is placed between the bridge's two endpoints.

Upon completion of the select bridge algorithm, a straight pillar is dropped down from each point that remains in the active point set. For added print stability, a 'footprint' is added for any straight pillar greater than 3mm in height from the print bed. A raft is also added to the bottom of each pillar for increased adhesion to the print bed. As in Dumas, et al. and Vanek, et al., the tips of the slanted pillars and straight pillars are tapered to a finer point for easier removal and post processing [2], [6].

Analysis

We tested our method on four models: a low-polygon count fish, Tiny T-Rex, a scan of an Aphrodite statue, and a scan of Michelangelo's David statue. The T-Rex, Aphrodite, and David models were sourced from Thingiverse.com and the fish was designed by the author. Each model requires a support structure to print successfully.

The method of determining points to support is slow. It iterates through every pixel in each layer. Since our grid is an n-pixel by n-pixel square that represents our printer's print bed, this results in an $O(n^2)$ asymptotic for our algorithm. However, in Table 1, we see that the majority of the time spent in getting the points to support is parsing through the gcode. As the alternate method of using surface normal vectors to determine points to support has an asymptotic of $O(n)$ where n is the number of facets, we gain nothing in performance by using the Boolean difference between layers.

Model	Number of Layers	Total Time to Get Points (ms)	Time to Parse Gcode	Time to Compare Layers
T-Rex	201	10576	4304	526
Fish	301	14984	3393	793
Aphrodite	751	58418	23251	5560
David	876	59463	25443	7090

Table 1 Time to Get Points

The method of generating the scaffolding runs in $O(n^2)$ as well, but in this case, our n is the number of points that require support, which is far less than the number of pixels as in the previous algorithm. In this case, each point that requires support has its anchoring segments created and is checked for intersections with the anchoring segments for every other point. As in practice, not all points will have anchoring segments that intersect, we can run in less than $O(n^2)$ time. The number of iterations for the support generation algorithm is also dependent on the number of layers in the model, and so models with more layers do take more time to run.

Model	Number of Layers	Number of Points to Support	Time to Generate Scaffolding (ms)
T-Rex	201	21	47
Fish	301	44	309
Aphrodite	751	15	392
David	876	15	1104

Table 2 Time to Generate Scaffolding

Both our simple pillar support structure and our scaffolding support structures use less filament than Slic3r's automatically generated supports. The simple pillars use 36% to 81% of the filament that Slic3r's supports use, and the scaffolding supports use 23% to 52% of the filament that Slic3r's supports use. The models with our support structures also print in a comparable amount of time, from 96% of the time to print Slic3r's supports to 103% of the time to print Slic3r's supports.

Support Type	Filament for Supports (mm)	% of Slic3r Support Filament	Time to Print (minutes)	% of Slic3r Time to Print
Slic3r	3437	100%	114	100%
Simple Pillars	3264	60.43%	113	99.12%
Scaffolding	3215	49.39%	112	98.25%

Table 3 T-Rex Model Data

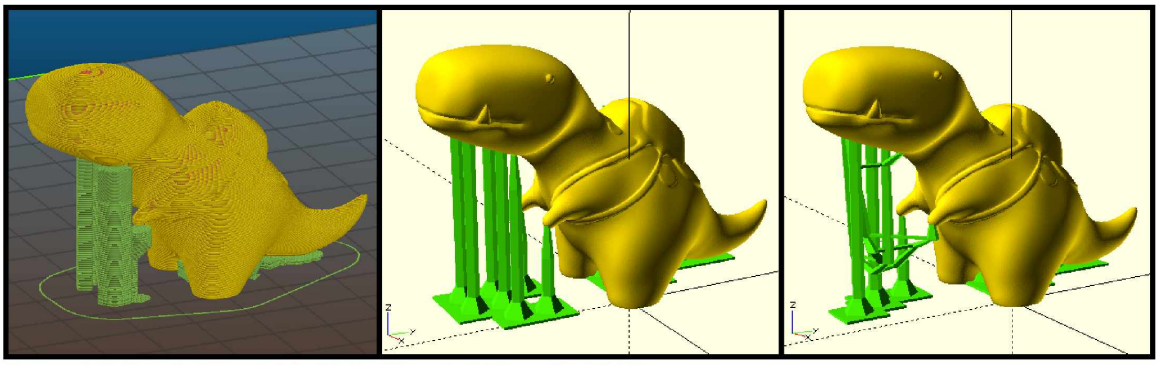

Figure 4 T-Rex: left to right: Slic3r, Simple Pillars, Scaffolding

Support Type	Filament for Supports (mm)	% of Slic3r Support Filament	Time to Print (minutes)	% of Slic3r Time to Print
Slic3r	2180	100%	168	100%
Simple Pillars	943	43.23%	173	102.98%
Scaffolding	751	34.43%	169	100.60%

Table 4 Fish Model Data

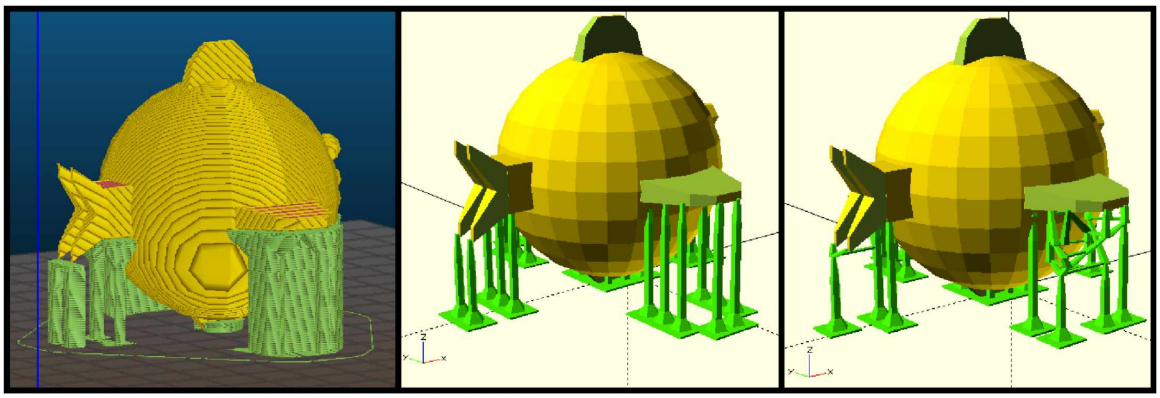

Figure 5 Fish: left to right: Slic3r, Simple Pillars, Scaffolding

Support Type	Filament for Supports (mm)	% of Slic3r Support Filament	Time to Print (minutes)	% of Slic3r Time to Print
Slic3r	1267	100%	445	100%
Simple Pillars	1034	81.61%	448	100.67%
Scaffolding	667	52.66%	446	100.22%

Table 5 Aphrodite Model Data

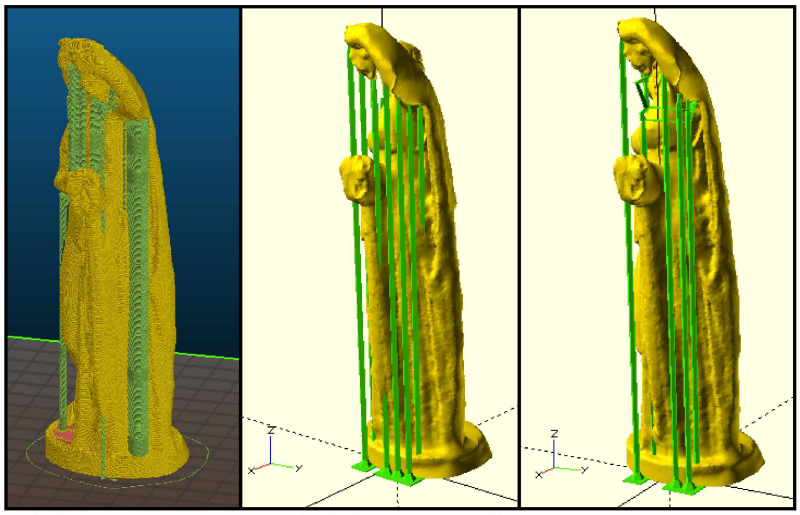

Figure 6 Aphrodite: left to right: Slic3r, Simple Pillars, Scaffolding

Support Type	Filament for Supports (mm)	% of Slic3r Support Filament	Time to Print (minutes)	% of Slic3r Time to Print
Slic3r	3094	100%	540	100%
Simple Pillars	1115	36.06%	532	98.52%
Scaffolding	715	23.10%	520	96.30%

Table 6 David Model Data

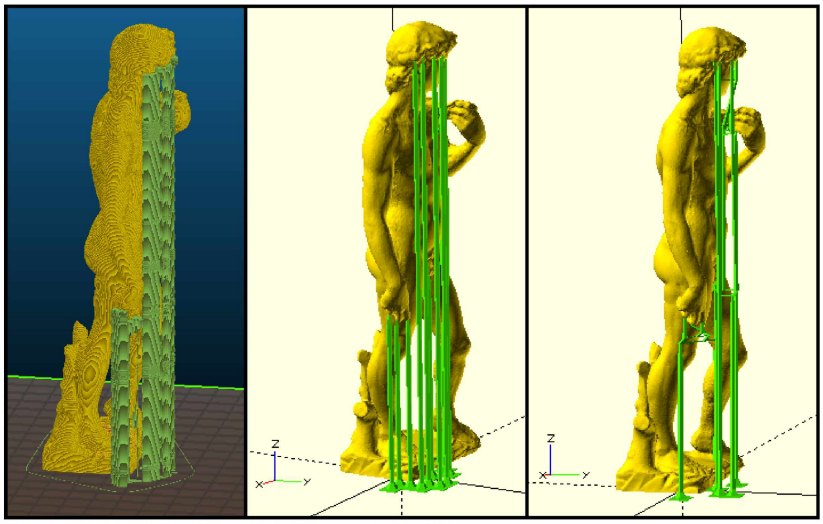

Figure 7 David: left to right: Slic3r, Simple Pillars, Scaffolding

We have printed the models with the supports we have generated. The T-Rex and fish printed successfully and the visual quality of the prints are comparable across support methods. However, the long support pillars experienced heavy deflection during the printing process for both the Aphrodite and David prints. The Aphrodite print was minimally affected by such deflections, but the elbow of the David print suffered and is offset in the printed model.

The post processing of the simple pillars and the scaffolding support is somewhat easier than removing the supports generated by Slic3r. In many cases, the supports separated from the model when it was removed from the print bed. Those that did not can be cut off with a pair of flush-cutters. The supports generated by Slic3r are more complex to remove, despite the thin interface layer between the supports and the model, and require the use of an X-acto knife.

Figure 8 Printed Fish: left to right: Slic3r, Simple Pillars, Scaffolding. (The Simple Pillars print was done before the square raft was added. Both the Simple Pillars and Scaffolding fish were printed before the final down-sampling ratio was determined for the selection of support points.)

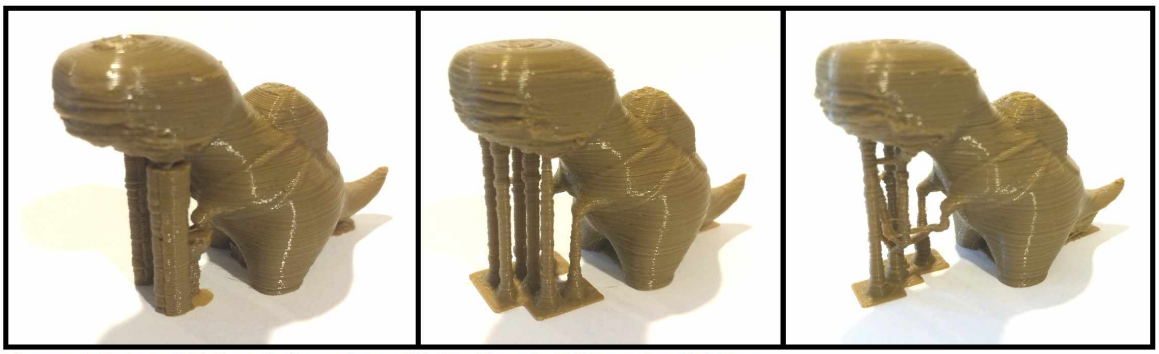

Figure 9 Printed T-Rex: left to right: Slic3r, Simple Pillars, Scaffolding

Figure 10 Aphrodite (left) and David (right). Note the misalignment of the Aphrodite print on her right hand holding up the garment and the misalignment of David's right elbow. These are due to the deflection of the tall support pillars during the printing process. Both models eventually recovered.

Conclusion and Future Work

Our method of both simple pillars and scaffolding supports both use less material and print in a similar amount of time compared to the supports automatically generated by Slic3r, which makes these methods preferable to Slic3r's supports when material is to be conserved. For models with shorter support columns, our method of scaffolding supports is ideal. The OpenSCAD rendering process is time-consumptive. Future work should explore alternate methods of unioning the model with the support structure.

However, because of the deflection of the long support pillars during the print process which can result in print failure, future work would be to determine a maximum pillar height and then add in a required bridge either between support pillars or between the support pillar and the model itself, to increase the pillar's stability. A change in the geometry of the long support pillars should also be examined in future work. If the pillars were wider at the base, at some proportion of the pillar's height, and tapered gradually to a point at the top in a very elongated cone, deflection may not be such an important problem.

Additional experimentation with the maximum horizontal distance and minimum vertical distance of points from the bridge should be run to determine the distances at which points offset from the bridge are no longer supported (See Appendix: Select Bridge Function).

Opportunities for future work exist in the parallelization of both the selection of points to be supported and the generation of scaffolding. Since each layer is only compared to the one below it, the 2D vector of points generated by parsing through the gcode could be broken up into multiple sections, with the layers at the borders between sections duplicated. The code could also be modified to run on the GPU. The generation of scaffolding could also be parallelized by having each thread work on a different sweep plane to select that plane's best bridge and then compare amongst themselves to determine the best bridge for that iteration of the algorithm.

Creating an interface that allows the user to set custom parameters for nozzle diameter, layer height, grid resolution, and maximum bridge length would give this tool a wider breadth of use, as would a graphical user interface.

Additionally, future work would be in selecting the points to be supported using the other method mentioned previously: by looking at the normal vectors of the STL file triangles. This would create a single tool that would not require the user to slice the model twice, first to generate the points that need support and then again to slice it after the supports have been generated.

References

[1] K. G. Cooper, *Rapid Prototyping Technology: Selection and Application*. New York, NY: Marcel Dekker, Inc., 2001.

[2] J. Dumas, J. Hergel, S. Lefebvre, "Bridging the Gap: Automated Steady Scaffoldings for 3D Printing," *ACM Trans. Graph*. 33, 4, Article 98, Jul. 2014.

[3] A. Hussein, et. al, "Advanced lattice support structures for metal additive manufacturing," *J Materials Process Technol*, vol. 213, pp. 1019-1026, Jul. 2013.

[4] P. F. Jacobs, *Rapid Prototyping & Manufacturing: Fundamentals of Stereolithography*. Dearborn, MI: Society of Manufacturing Engineers, 1992.

[5] G. Strano, et al, "A new approach to the design and optimisation of support structures in additive manufacturing," *Int J Adv Manuf Technol*, vol. 66, pp. 1247-1254, 2013.

[6] J. Vanek, et al, "Clever Support: Efficient Support Structure Generation for Digital Fabrication" *Computer Graphics Forum*, vol. 33, n. 5, pp. 108-125, 2014.

[7] I. Zein, et al, "Fused deposition modeling of novel scaffold architectures for tissue engineering applications," *Biomaterials*, vol. 23, pp. 1169-1185, 2002.

Models

3DWP, "Aphrodite Statue 3D Scan", thingiverse.com/thing:1409194 Mar. 11, 2016.

3DWP, "David By Michelangelo Sculpture (Statue 3D Scan)", thingiverse.com/thing:502967 Oct. 16, 2014.

3DWP, "Tiny T-Rex (Partially Hollow for Balance)", thingiverse.com/thing:615875 Dec. 30, 2014.

A. Tupek, "Low-Poly Fish", thingiverse.com/thing:1491074 Apr. 14, 2016.

Appendix

Main Function from Generate Scaffolding

```
int main()
{
  //insert header info to scad file
  setup_scad();

  set<Point> points_for_alg3;
  set<Anchoring_Segment> segments_for_alg3;
  vector<Sweep_line> sweep_line_vec;

  set<Point> active_points = get_pts_from_file();
  int num_points = active_points.size();

  make_sweep_vector();

  while(true)
  {
    Bridge the_best_bridge;
    for(auto outer_loop_index = 0;
          outer_loop_index < slope_of_sweep.size();
          outer_loop_index++)
    {
      segments.clear();
      create_anchoring_segments(active_points,
          active_bridges, segments, slope_of_sweep,
          outer_loop_index);

      active_events.clear();
      create_events(segments, active_events);

      //find intersections between sweep plane and
          anchoring segments at each event
      //these intersections are the points sent to
          select bridge
      sweep_line_vec.clear();
      find_intersections(active_events, slope_of_sweep,
          outer_loop_index, sweep_line_vec);

      sort_sweep_lines(sweep_line_vec,
          slope_of_sweep[outer_loop_index]);

      Bridge temp_bridge;
      temp_bridge = select_bridge(sweep_line_vec,
          slope_of_sweep[outer_loop_index]);
      if(temp_bridge.score > the_best_bridge.score)
      {
```

```
        the_best_bridge = temp_bridge;
      }

    }

    if(the_best_bridge.p1.x == 0
     && the_best_bridge.p1.y == 0
     && the_best_bridge.p1.z == 0)
    {
      break;
    }
    snap(the_best_bridge, active_points);
  }

  //drop pillars from all remaining points that need
     support
  send_remaining_points_to_scad(active_points);

  return 0;
}
```

Select Bridge Function

```
Bridge select_bridge(vector<Sweep_line> & sweep_lines,
     double sweep_slope)
{
  Bridge best_bridge;
  double max_distance = 30.0;
  double max_horizontal = 5.0;
  double min_vertical_dist = 2.0;
  double neg_inf(-std::numeric_limits<double>::infinity());
  double best_score = neg_inf;

  //put z-coords into set
  //sorted by increasing z
  set<double> z_set;
  for(auto i = sweep_lines.begin();
          i != sweep_lines.end(); i++)
  {
    for(auto j = i->intersected_points.begin();
          j != i->intersected_points.end(); j++)
    {
      z_set.insert(j->first.z);
    }
  }
```

```
//check for bridges at more than the z_heights where points
    need support
  double z_max;
  if(z_set.size() == 1)
  {
    z_max = *(z_set.begin());
  }
  else
  {
    z_max = *(--z_set.end());
  }

  for(auto i = 0.0; i < z_max; i += 1.0)
  {
    z_set.insert(i);
  }

  //iterate through each sweep line
  for(auto i = sweep_lines.begin();
          i != sweep_lines.end(); i++)
  {
    //do this for each z-level in the z-set
    for(auto j = z_set.begin(); j != z_set.end(); j++)
    {
     /*sweep_line has vector of intersections, go through
     and for every pair of intersections (m, n)
     check if points are within max_distance
     if they are, add endpts at j's z-value (endp1 = m,
     endp2 =n)
     check if points between (m, n inclusive) are above j's
     z-value
     if they are, add points to bridge's supported points
     vector
     get bridge's gain & score
     then check pair of intersections (m+1, n)*/
      for(int m = 0; m < i->intersected_points.size()-1;
          m++)
      {
        for(int n = m+1; n < i-> intersected_points.size();
          n++)
        {
          Bridge temp_bridge
          (i->intersected_points[m].first,
               i->intersected_points[n].first, *j);
          temp_bridge.p1.z = *j;
          temp_bridge.p2.z = *j;
```

```
if((temp_bridge.length < max_distance) &&
(temp_bridge.length >= 5))
{
  //add m, n and any points between them to
     bridge's supported points vector
  for(int k = m; k < n+1; k++)
  {
  //first check constraints for angled pillar...
    double temp_horiz_dist = calc_horiz_dist
          (temp_bridge.p1, temp_bridge.p2,
               i->intersected_points
               [k].second.endpt1);
    double temp_vert_dist = calc_z_diff
          (i->intersected_points
          [k].second.endpt1, temp_bridge.height);
    if((i->intersected_points
          [k].second.endpt1.z >= *j) &&
          (temp_horiz_dist <= max_horizontal)&&
          (temp_vert_dist >= min_vertical_dist))
     {
        temp_bridge.supported_points.insert
          (i->intersected_points
          [k].second.endpt1);
     }
   }
    //and now we calculate bridge's gain
    double temp_gain = calculate_gain
          (*j, temp_bridge.length,
          temp_bridge.supported_points.size());
    //if gain > 0, calculate l_max & score
    if(temp_gain > 0)
    {
      double temp_lmax = calculate_lmax
          (temp_bridge.p1, temp_bridge.p2,
          temp_bridge.supported_points);
      double temp_score = calculate_score
          (temp_gain,
          temp_bridge.supported_points.size(),
          temp_lmax);
      temp_bridge.score = temp_score;
      if(temp_score > best_score)
      {
        best_score = temp_score;
        best_bridge = temp_bridge;
      }
    }
  }
}
```

```
        }
      }
    }
  return best_bridge;
}
```

Snap Function

```
void snap(Bridge & best_bridge, set<Point> & active_pts)
{
  //remove supported points from set of points that need
     support (active_pts)
  for(auto i = best_bridge.supported_points.begin();
     i != best_bridge.supported_points.end(); i++)
  {
    remove_supported_pt_from_active_set(*i, active_pts);
  }

  //bridge comments for scad outfile
  out_file << "//STARTING NEW BRIDGE DATA" << endl;
  out_file << "//bridge data:" << endl;
  out_file << "/*" <<endl;
  best_bridge.print_bridge_members(out_file);
  out_file << "//supporting points:" << endl;
  for(auto i = best_bridge.supported_points.begin();
     i != best_bridge.supported_points.end(); i++)
  {
    i->print_coords_with_z(out_file);
  }
  out_file << "*/" <<endl;

  //drop slanted pillar for each supported point
  for(auto i = best_bridge.supported_points.begin();
     i != best_bridge.supported_points.end(); i++)
  {

    //pillar(x, y, height of pillar base,
     vertical height of pillar)
    //slanted_pillar(x1, y1, z1, x2, y2, z2)
    //where x1, y1, z1 are coords of pt that needs support
    //and x2, y2, z2 are coords of closest point on bridge
     to pt that needs support
    Point point_on_bridge = find_closest
          (best_bridge.p1, best_bridge.p2, *i);
    out_file << "//slanted pillar" << endl;
    out_file << "\tslanted_pillar(" << i->x << ", " <<
          i->y << ", " << i->z << ", " << point_on_bridge.x
```

```
            << ", " << point_on_bridge.y << ", " <<
            best_bridge.height << ");" << endl;
  }

  //lay bar from bridge endpt1 to endpt2
  //and add pts to active_pts
  //first check for horizontal
  //if horizontal, use bridge1
  if(best_bridge.p1.y == best_bridge.p2.y)
  {
    out_file << "//bridge:" << endl;
    out_file << "\tbridge1(" << best_bridge.p1.x << ", " <<
          best_bridge.p1.y << ", " << best_bridge.p2.x <<
          ", " << best_bridge.p2.y << ", " <<
          best_bridge.height << ");" << endl;
  }
  else //otherwise use bridge
  {
    out_file << "//bridge:" << endl;
    out_file << "\tbridge(" << best_bridge.p1.x << ", " <<
          best_bridge.p1.y << ", " << best_bridge.p2.x <<
          ", " << best_bridge.p2.y << ", " <<
          best_bridge.height << ");" << endl;
  }

  if(best_bridge.p1.x != 0 && best_bridge.p1.y != 0
     && best_bridge.p1.z != 0)
  {
    active_pts.insert(best_bridge.p1);
  }

  if(best_bridge.p2.x != 0 && best_bridge.p2.y != 0 &&
     best_bridge.p2.z != 0)
  {
    active_pts.insert(best_bridge.p2);
  }
}
```

Send Remaining Points to OpenSCAD Function

```
void send_remaining_points_to_scad(set<Point> &active_pts)
{
  out_file << "//remaining active points, dropping pillars
     to surface" << endl;
  for(auto i = active_pts.begin();
     i != active_pts.end(); i++)
  {
    out_file << "\tpillar(" << i->x << ", " << i->y << ",
```

```
      0, " << i->z << ");" << endl;
    if(i->z > 3.0)
    {
      out_file << "\tfoot(" << i->x << ", " << i->y << ");"
          << endl;
    }
    out_file << "\traft(" << i->x << ", " << i->y << ");"
     << endl;
  }
  //and output final closing brace for the union of all
     supports
  out_file << "}" << endl;
}
```

OpenSCAD Setup Function

```
void setup_scad()
{
  out_file << "//scad pillars and bars generated by code"
     << endl << endl;
  out_file << "$fa = 5;" << endl;
  out_file << "$fn = 5;" << endl;
  out_file << "radius = 1.0;" << endl;
  out_file << "inner = 0.5;" << endl;
  out_file << "delta = 0.5;" << endl;
  out_file << "height = 0.4;" << endl;
  out_file << "foot_radius = 2.5;" << endl;
  out_file << "foot_height = 2.5;" << endl;
  out_file << "raft_width = 7;" << endl;
  out_file << "raft_height = 0.4;" << endl;
  out_file << "epsilon = 0.01;" << endl << endl;
  out_file << "module raft(x_coord, y_coord)" << endl;
  out_file << "{" << endl;
  out_file << "\ttranslate([x_coord, y_coord, 0])" << endl;
  out_file << "\tcube([raft_width, raft_width,
     raft_height], center = true);" << endl;
  out_file << "}" << endl;
  out_file << "module foot(x_coord, y_coord)" << endl;
  out_file << "{" << endl;
  out_file << "\ttranslate([x_coord, y_coord, 0])" << endl;
  out_file << "\tdifference()" << endl;
  out_file << "\t{" << endl;
  out_file << "\t\tcylinder(foot_height, foot_radius,
     radius);" << endl;
  out_file << "\t\ttranslate([0, 0, -epsilon])" << endl;
  out_file << "\t\tcylinder(foot_height+2*epsilon,
     foot_radius-inner, radius-inner);" << endl;
  out_file << "\t}" << endl;
```

```
out_file << "}" << endl;
out_file << "module circle1(x_coord, y_coord, z_coord)"
   << endl;
out_file << "{" << endl;
out_file << "\ttranslate([x_coord, y_coord, z_coord])"
   << endl;
out_file << "\tcylinder(.2, radius-.5, radius-.5);" <<
   endl;
out_file << "}" << endl;
out_file << "module circle2(x_coord, y_coord, z_coord)"
   << endl;
out_file << "{" << endl;
out_file << "\ttranslate([x_coord, y_coord, z_coord])" <<
   endl;
out_file << "\tcylinder(.2, radius, radius);" << endl;
out_file << "}" << endl;
out_file << "module slanted_pillar(x_coord1, y_coord1,
   z_coord1, x_coord2, y_coord2, z_coord2)" << endl;
out_file << "{" << endl;
out_file << "\thull()" << endl;
out_file << "\t{" << endl;
out_file << "\t\tcircle1(x_coord1, y_coord1, z_coord1);"
   << endl;
out_file << "\t\tcircle2(x_coord2, y_coord2, z_coord2);"
   << endl;
out_file << "\t}" << endl;
out_file << "}" << endl;
out_file << "module pillar(x_coord, y_coord, z_coord,
   z_height)" << endl;
out_file << "{" << endl;
out_file << "\tif(z_height > 5)" << endl;
out_file << "\t{" << endl;
out_file << "\t\thull()" << endl;
out_file << "\t\t{" << endl;
out_file << "\t\t\tcircle1(x_coord, y_coord, z_height);"
   << endl;
out_file << "\t\t\tcircle2(x_coord, y_coord,
   z_height-5);" << endl;
out_file << "\t\t}" << endl;
out_file << "\t\ttranslate([x_coord, y_coord, z_coord])"
   << endl;
out_file << "\t\t{" << endl;
out_file << "\t\t\tdifference()" << endl;
out_file << "\t\t\t{" << endl;
out_file << "\t\t\t\tlinear_extrude(z_height-5+epsilon)"
   << endl;
out_file << "\t\t\t\t{" << endl;
out_file << "\t\t\t\t\tcircle(radius);" << endl;
```

```
out_file << "\t\t\t\t}" << endl;
out_file << "\t\t\t\ttranslate([0, 0, -epsilon])" <<
   endl;
out_file << "\t\t\t\tlinear_extrude(z_height-5)" << endl;
out_file << "\t\t\t\t{" << endl;
out_file << "\t\t\t\t\tcircle(radius-inner);" << endl;
out_file << "\t\t\t\t}" << endl;
out_file << "\t\t\t}" << endl;
out_file << "\t\t}" << endl;
out_file << "\t}" << endl;
out_file << "\telse" << endl;
out_file << "\t{" << endl;
out_file << "\t\ttranslate([x_coord, y_coord, z_coord])"
   << endl;
out_file << "\t\t{" << endl;
out_file << "\t\t\tdifference()" << endl;
out_file << "\t\t\t{" << endl;
out_file << "\t\t\t\tlinear_extrude(z_height)" << endl;
out_file << "\t\t\t\t{" << endl;
out_file << "\t\t\t\t\tcircle(radius);" << endl;
out_file << "\t\t\t\t}" << endl;
out_file << "\t\t\t\ttranslate([0, 0, -epsilon])" <<
   endl;
out_file << "\t\t\t\tlinear_extrude(z_height+2*epsilon)"
   << endl;
out_file << "\t\t\t\t{" << endl;
out_file << "\t\t\t\t\tcircle(radius-inner);" << endl;
out_file << "\t\t\t\t}" << endl;
out_file << "\t\t\t}" << endl;
out_file << "\t\t}" << endl;
out_file << "\t}" << endl;
out_file << "}" << endl;
out_file << "//does not work for horizontal bridges, use
   bridge1" << endl;
out_file << "module bridge(x0, y0, x1, y1, z_height)" <<
   endl;
out_file << "{" << endl;
out_file << "translate([0, 0, z_height])" << endl;
out_file << "linear_extrude(height)" << endl;
out_file << "polygon(points = [[x0+delta, y0],
   [x0-delta, y0], [x1-delta, y1], [x1+delta, y1]]);" <<
   endl;
out_file << "}" << endl << endl;
out_file << "//does not work for vertical bridges, use
   bridge" << endl;
out_file << "module bridge1(x0, y0, x1, y1, z_height)" <<
   endl;
out_file << "{" << endl;
```

```
  out_file << "translate([0, 0, z_height])" << endl;
  out_file << "linear_extrude(height)" << endl;
  out_file << "polygon(points = [[x0, y0+delta],
     [x0, y0-delta], [x1, y1-delta], [x1, y1+delta]]);" <<
     endl;
  out_file << "}" << endl << endl;
  out_file << "union()" << endl;
  out_file << "{" << endl; //union opening brace
}
```

OpenSCAD Example File

```
$fa = 5;
$fn = 5;
radius = 1.0;
inner = 0.5;
delta = 0.5;
height = 0.4;
foot_radius = 2.5;
foot_height = 2.5;
raft_width = 7;
raft_height = 0.4;
epsilon = 0.01;

module raft(x_coord, y_coord)
{
  translate([x_coord, y_coord, 0])
  cube([raft_width, raft_width, raft_height],
     center = true);
}
module foot(x_coord, y_coord)
{
  translate([x_coord, y_coord, 0])
  difference()
  {
    cylinder(foot_height, foot_radius, radius);
    translate([0, 0, -epsilon])
    cylinder(foot_height+2*epsilon, foot_radius-inner,
     radius-inner);
  }
}
module circle1(x_coord, y_coord, z_coord)
{
  translate([x_coord, y_coord, z_coord])
  cylinder(.2, radius-.5, radius-.5);
}
module circle2(x_coord, y_coord, z_coord)
{
```

```
  translate([x_coord, y_coord, z_coord])
  cylinder(.2, radius, radius);
}
module slanted_pillar(x_coord1, y_coord1, z_coord1,
     x_coord2, y_coord2, z_coord2)
{
  hull()
  {
    circle1(x_coord1, y_coord1, z_coord1);
    circle2(x_coord2, y_coord2, z_coord2);
  }
}
module pillar(x_coord, y_coord, z_coord, z_height)
{
  if(z_height > 5)
  {
    hull()
    {
      circle1(x_coord, y_coord, z_height);
      circle2(x_coord, y_coord, z_height-5);
    }
    translate([x_coord, y_coord, z_coord])
    {
      difference()
      {
        linear_extrude(z_height-5+epsilon)
        {
          circle(radius);
        }
        translate([0, 0, -epsilon])
        linear_extrude(z_height-5)
        {
          circle(radius-inner);
        }
      }
    }
  }
  else
  {
    translate([x_coord, y_coord, z_coord])
    {
      difference()
      {
        linear_extrude(z_height)
        {
          circle(radius);
        }
        translate([0, 0, -epsilon])
```

```
        linear_extrude(z_height+2*epsilon)
        {
          circle(radius-inner);
        }
      }
    }
  }
}
//does not work for horizontal bridges, use bridge1
module bridge(x0, y0, x1, y1, z_height)
{
  translate([0, 0, z_height])
  linear_extrude(height)
  polygon(points = [[x0+delta, y0], [x0-delta, y0],
     [x1-delta, y1], [x1+delta, y1]]);
}

//does not work for vertical bridges, use bridge
module bridge1(x0, y0, x1, y1, z_height)
{
  translate([0, 0, z_height])
  linear_extrude(height)
  polygon(points = [[x0, y0+delta], [x0, y0-delta],
     [x1, y1-delta], [x1, y1+delta]]);
}
union()
{
  translate([18, 24, 17])
  {
    color("Orchid", 0.5)
    sphere(0.75, center = true);
  }
  translate([11, 15, 20])
  {
    color("Orchid", 0.5)
    sphere(0.75, center = true);
  }
  translate([19, 9, 15])
  {
    color("Orchid", 0.5)
    sphere(0.75, center = true);
  }
  translate([25, 20, 18])
  {
    color("Orchid", 0.5)
    sphere(0.75, center = true);
  }
```

```
  //with structure generated from test points
  //STARTING NEW BRIDGE DATA
  //bridge data:
  /*
  Point1: 11, 20
  Point2: 25, 20
  Bridge length: 14
  Bridge height: 15
  P1 open: 1
  P2 open: 1
  //supporting points:
  25, 20, 18
  18, 24, 17
  11, 15, 20
  */
  //slanted pillar
  slanted_pillar(25, 20, 18, 25, 20, 15);
  //slanted pillar
  slanted_pillar(18, 24, 17, 18, 20, 15);
  //slanted pillar
  slanted_pillar(11, 15, 20, 11, 20, 15);
  //bridge:
  bridge1(11, 20, 25, 20, 15);
  //remaining active points, dropping pillars to surface
  pillar(11, 20, 0, 15);
  foot(11, 20);
  raft(11, 20);
  pillar(19, 9, 0, 15);
  foot(19, 9);
  raft(19, 9);
  pillar(25, 20, 0, 15);
  foot(25, 20);
  raft(25, 20);
}
```

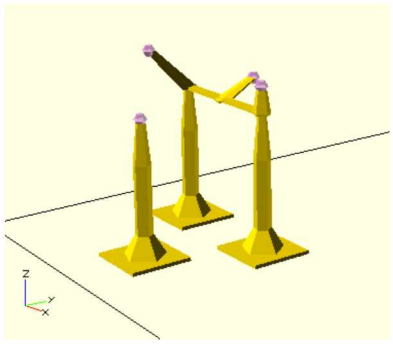

Figure 11 Example OpenSCAD Output. Note that the pink dots are the points that need to be supported. Showing them in the OpenSCAD model is optional for the user.